

Flasher

series

SIGNUM SYSTEMS CORPORATION

Command-Line Flasher for ARM, XScale and Cortex

User Manual

SIGNUM
S Y S T E M S

COPYRIGHT NOTICE

Copyright (c) 2011 by Signum Systems Corporation. All rights are reserved worldwide. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Signum Systems.

DISCLAIMER

Signum Systems makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Also, Signum Systems reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Signum Systems to notify any person or organization of such revision or changes.

WARRANTY

Signum Systems warrants to the original purchaser that this product is free of defects in material and workmanship and performs to applicable published Signum Systems specifications for a period of SIX MONTHS from the date of shipment. If defective, the product must be returned to Signum Systems, prepaid, within the warranty period, and it will be repaired or replaced (at our option) at no charge. Equipment or parts which have been subject to misuse, abuse, alteration, neglect, accident, unauthorized installation or repair are not covered by warranty. This warranty is in lieu of any other warranty expressed or implied. IN NO EVENT SHALL SIGNUM SYSTEMS BE LIABLE FOR CONSEQUENTIAL DAMAGES OF ANY KIND. It is up to the purchaser to determine the reliability and suitability of this product for his particular application.

SIGNUM
S Y S T E M S
1211 FLYNN RD., UNIT #104
CAMARILLO, CA 93012, U.S.A
PHONE 805 • 383 • 3682
WWW.SIGNUM.COM

Table of Contents

Introduction	1
License	1
Installation	2
Installation Directory Content	2
First Time Run	2
Usage	3
Emulation Parameter File	5
Macro/Initialization Files	5
Programming without Target RAM	6
Error Codes	6
Board and CPU Customization	8
Supported Flash Devices	8
Programming External CFI NOR Flash	9
Programming Internal STR9 Flash	10
Programming Internal LPC Flash	10
Programming Internal LM3S Flash	15
Programming Internal SAM3 FLASH	15
Programming Internal STM32 FLASH	18
Programming Internal TPM3 FLASH	18
Programming NAND Flash	20
Programming SPI NOR flash	23
The RAM Pseudo-Programmer	25
Programming Multiple Devices	25
Custom Flash Devices	26

The logo consists of a light gray square. Inside the square, the word "Flasher" is written in a bold, black, sans-serif font at the top. At the bottom, the word "utility" is written in a white, lowercase, sans-serif font.

Flasher is a command-line oriented flash programming tool for the Signum JTAGjet emulator intended to aid the automation of your production process.

Introduction

This document describes the Flasher package designed to program flash devices from the command-line window (DOS prompt) or from within your application. Since no popup messages are displayed during the programming process, Flasher lends itself to use as a component of complex, fully automated production tool packages.

License

Flasher is designed for the use with the Signum JTAGjet emulator. This flasher-emulator tandem combination requires a license. If your emulator does not have it, an external license file must be obtained from Signum Systems. Once received, the Jxxxxx-Flasher.lic file needs to be placed in the directory where SigFlashCmd.exe resides. (The xxxxx part of the actual file name is the serial number of your JTAGjet emulator). Alternatively, the license file can be placed in the C:\Signum\Licenses folder where it becomes available to other applications, such as Chameleon or Flasher.

Installation

To install the Flasher package, execute the setup_flasher.exe program. The default installation folder is C:\Signum\Flasher.

Note: White spaces in file and folder names may interfere with DOS command-line programming and thus should be avoided.

If system-level JTAGjet USB driver has not been installed on your computer earlier, the system will prompt you for the appropriate driver when the emulator is connected to the USB port for the first time. Point the installation program to the Drivers\USB subfolder of the installation folder (in our example, C:\Signum\Flasher\Drivers\USB.)

Installation Directory Content

Table 1 details the contents of the Flasher installation directory.

DELIVERABLE	DESCRIPTION
SigFlashCmd.exe	Command line executable program. Also called from within t.bat.
.\bin*.*	Flasher executable components.
.\docs\docs.htm	Main documentation accessible through a shortcut after installation.
.\docs\flasher_um.pdf	This document.
.\docs*.*	Additional documentation (accessible via docs.htm file).
.\boards*.par, *.ini	CPU and target specific parameter and board initialization files.
p*.bat	Programming scripts for selected evaluation boards.
.\Drivers\USB*.*	JTAGjet USB driver.
.\tests*.*	Test scripts and log files from test runs.
.\testdata*.*	Test data used by test scripts.
t.bat	Test execution script. Example: t_tinnov1510.

TABLE 1 The contents of installation directory.

First Time Run

Make sure that your emulator is connected to a USB port. A dialog box will prompt you to select the JTAGjet emulator to be used to program the flash. The information, saved in the SigFlashCmd.ini file, will be used in subsequent sessions. Connecting to another emulator is possible by using the -connect switch or by deleting the SigFlashCmd.ini file. The -connect option allows you to specify the emulator serial number. The -connect option overrides the counterpart setting in the SigFlashCmd.ini file.

Usage

SigFlashCmd.exe displays usage information if executed with the -help option (or without any options):

```
SigFlashCmd Version 1.15 (C) Signum Systems Corp. 2006-2009
Usage:
  SigFlashCmd [options]

Available options:

-help                - Display help and all options if used with -flash.
-v                  - Display Flasher's version - can be used with -flash.
-connect            - Specify connection via a dialog box.
-connect <sn>       - Connect to the JTAGjet with serial number <sn>.

-cpu <cputype>      - The CPU core type: ARM7TDMI, ARM926EJ-S, C166 etc.
                    - The <cputype>.par file will be used if it exists.
-nocpu              - Ignore any CPU connection errors and continue.
-emu <file> ...     - Set emulation parameters from <file>. CPU dependent.
-emu <n>=<v> ...     - Set emulation parameter <n> to <v>. CPU dependent.

-init <file> ...    - The initialization script file. Board dependent.
-term <file> ...    - The termination script file. Board dependent.

-flash <type>@<addr>[,<opt> ...] - The flash type (CFI, STR9, NAND etc.),
                    - base address, and comma-separated flash options.

-ram <size>@<addr> - The RAM range accessible to the programmer.
-dump               - Dump the flash geometry details.

-erase chip         - Erase the chip.
-erase <range>      - Erase all the sectors in the range.
-erase              - Erase all programmed ranges before programming.

-blankcheck chip    - Blank check the entire chip.
-blankcheck <range> - Blank check the range.
-blankcheck         - Blank check only the erased or programmed ranges.

-format <format>    - The format of the flash image file. The following
                    - file formats are accepted:
                        bin - Plain binary file (default).
                        hex - Intel HEX format.
                        srec - Motorola S-record format.
                        Msbin - Windows CE binary image data format.
-offset [<addr>]    - The address offset for data from a file. If <addr>
                    - is missing, use the <addr> from the -flash option.

-program <file>     - Program the flash with an image file.
-range              - Ignore all data outside flash.
-range <range>      - Ignore all data outside a given range.
-dumpblk            - Display all code ranges (after applying -range).
```

```
-verify          - Verify the flash image after programming.
-verify <file>   - Verify the flash using a file without programming.

-extra <cmd>     - Execute an extra command. Flash type dependent.
                  More than one -extra options can be given.
-extra          - List all extra commands for the flash in use.

-read <range>    - Read and display the data from the given range.
-save <fname>    - Save the read data to a binary file w/o display.

-q             - Quiet mode. Display only the results and errors.
-time          - Display the execution time for each step.

<range>: <size>@<addr> - <size> in bytes; <addr> is physical address.
    Sn|Bn          - A single sector or block. (S1, B0 are first.)
    Sn..Sm|Bn..Bm - Range of sectors or blocks. (Sn <= Sm, Bn <= Bm.)
    <size>@Sn|Bn   - <size> bytes starting at sector Sn or block Bn.
```

If the -help and -flash options are used at the same time (as in “-help -flash CFI”), the help information includes a list of the flash programmer options for the specified programmer.

All addresses and sizes can be specified as one of the following:

- Decimal number (e.g., 1024)
- Hexadecimal number (e.g., 0x400)
- Hexadecimal or decimal number with K or M appended (e.g., 512K or 4M)

For testing purposes, the -program and -verify options accept their parameters in the <size>@<addr> format instead of a file name (<fname>). This results in using the testing pattern 0,1,2,3,...,0xFF, 1,2,3,0xFF,0, 2,3, ... of the specified size, which is useful for validating the configuration and the efficiency of operations of the flash device.

Examples

Note: The backslash character (\) in the examples below symbolizes line continuation—the entire command should be typed on one line. Please refer to the t.bat file for more usage examples of SigFlashCmd usage.

QUERY THE FLASH CAPABILITIES

```
SigFlashCmd -emu OMAP1710.par -flash cfi@0 -dump
```

PROGRAM A BIN FILE INTO THE CFI FLASH AT ADDRESS 0

```
SigFlashCmd -emu omap1710.par -ram 1K@0x20000000 -flash cfi@0 \
-erase -program test.bin -verify
```

VERIFY A BIN FILE INTO THE CFI FLASH AT ADDRESS 0

```
SigFlashCmd -emu omap1710.par -ram 1K@0x20000000 -flash cfi@0 \
-verify test.bin
```

PROGRAMMING A TEST PATTERN INTO THE FLASH WITHOUT VERIFICATION

```
SigFlashCmd -emu omap1710.par -ram 1K@0x20000000 -flash cfi@0 \
-erase -program 1M@0x10000
```

Emulation Parameter File

The core and CPU dependent emulation parameters are stored in the .par file used in conjunction with the -emu option. These parameters are a subset of the parameter list displayed by Chameleon's emu command. Refer to the .par files provided with the package for guidelines in determining which of these parameters are essential to the operation of the programmer. If found in the programmer package, the .par file designated for your target can be used unmodified.

You can use multiple -emu file options and set individual emulation parameters using the “-emu name=value” syntax.

Macro/Initialization Files

The .mac or .ini files, used in conjunction with the -init and -term options allow you to utilize the following subset of Chameleon Debugger's macro commands.

C O M M A N D	D E S C R I P T I O N
S8/SB/S16/SW/S32/SD <addr> = <value>	Set Byte, 16 bit Word, 32 bit Double word.
S8/SB/S16/SW/S32/SD <addr> = <value>	OR value with memory at <addr>
S8/SB/S16/SW/S32/SD <addr> &= <value>	AND value with memory at <addr>
S8/SB/S16/SW/S32/SD <addr> ^= <value>	XOR value with memory at <addr>
D8/DB/D16/DW/D32/DD <addr>	Reads Byte, 16 bit Word, 32 bit Double word. Displays them when ECHO is on.
ECHO ON OFF	ECHO control. OFF by default.
PAUSE <ms>	Pause the execution of commands for <ms> milliseconds. Use, for example, to allow the PLL to stabilize.
EMU <name> = <value>	Emulation parameters (the same as in Chameleon).

C O M M A N D	D E S C R I P T I O N
RESET [/HALT]	Reset the CPU and, optionally, stop the CPU if it did not stop automatically after a reset.
STOP	Stop the CPU.
MEM SPACE	Display the current memory space (in the echo mode only).
MEM SPACE DEFAULT	Select the default (index 0) memory space.
MEM SPACE <space-index>	Select the memory space defined by an index. E.g., 0x1F - CP15 register access; 0x1E - CP14 access.

TABLE 2 Chameleon macro commands that can be use with the -init and -term options.

Note: Avoid using Chameleon’s startup macros “as is.” Most Chameleon macros can be simplified for flash programming purposes.

Programming without Target RAM

Most flash devices can be programmed without using the RAM buffer on the target. If the -ram parameter is not specified, all programming actions are executed by accessing the flash or the flash controller directly through the JTAG. While the speed of the erase operation is the same as when running the code in the target’s RAM, all the other memory operations may slow down significantly. The RAM-less method is typically only suitable for small boot-loaders or in case of emergency board recovery. Therefore, specifying RAM on the target and configuring it is strongly recommended. Most programmers require no more than 2K of RAM. The best results are achieved with internal static RAM. Use an initialization macro in conjunction with the -init option to enable and configure access to RAM that is not accessible all the time.

It is also possible to setup TCM memory as RAM for programmer. It requires writes to CP15 registers from an INI file (by using MEM SPACE command). See the .\boards\dm644x_495.ini file for an example of TCM configuration sequence.

Some programmers for on-chip flash devices, such as the LPC programmer, do not require RAM to be specified and will automatically use the available on-chip RAM for programming purposes.

Error Codes

Table 3 shows Flasher error codes. These codes can be used by the IF ERRORLEVEL N command in DOS batch files. A detailed description of each error is displayed in the DOS window. A SigFlashCmd.exe run terminates with one of the two following messages appearing as the last line in the DOS window:

** Exiting with error code NN

- 0r -

OK - Done

ERROR CODE	DESCRIPTION	WHAT TO DO
NON-PROGRAMMING ERRORS		
0	All required actions executed successfully.	No action required.
1	Usage problem: bad command line options.	Correct the command line options.
2	Emulator connection problem: the emulator specified with the -connect option or in the SigFlashCmd.ini file not found.	Make sure that the emulator is connected and not used by a different application. Use the -connect option to list the available emulators and to connect to a new emulator. Make sure that your emulator has a license for an external programmer.
3	Emulator license error.	Make sure that the Jxxxxx-Flasher.lic file is in the same folder as SigFlashCmd.exe.
4	Initial CPU connection problem: no power, cannot stop the CPU, etc.	Verify the JTAG connection and check power on the target. Make sure that the file specified in the -emu option defines the JTAG geometry and sets emulation parameters correctly. Power cycle the board and restart the process. Try to connect to the CPU from Chameleon Debugger.
5	Initialization macro execution problem. (The -init option.)	Verify the command syntax. Check the macro using Chameleon Debugger. Note, however, that the HALT option cannot be used with the “reset” macro command.
6	Termination macro execution problem. (The -term option.)	Ditto
7-10	Reserved	
PROGRAMMING ERRORS		
11	Incorrect flash related parameters, such as a nonexistent flash programmer or unknown file format.	Details will be displayed in the window. Make sure that the command line options are set correctly.
12	Internal error — one of flash related DLLs missing.	Make sure that SigFlashCmd is started in the folder where it is located — check the “Start in” field in the shortcut to the program.
13	Flash initialization problem, such as “cannot detect CFI flash at the specified address.”	Make sure that the CPU is properly initialized.
14	Cannot open a file when programming and verifying.	Make sure that the file exists and the -format option specifies its format correctly.

E R R O R C O D E	D E S C R I P T I O N	W H A T T O D O
15	Failure while erasing.	Make sure that the CPU is initialized and that flash write enable is set.
16	Failure while blank-checking.	Make sure, that the chip or sectors are erased. Make sure that the CPU is initialized properly.
17	Failure while programming	Same as for erasing. (See code 15).
18	Failure while verifying the code.	Make sure that verification is performed on the file used to program the flash.

TABLE 3 SigFlashCmd error codes.

Board and CPU Customization

Programming flash devices on custom-made processors and boards involves the following steps.

1. Adjust the command line parameters: modify memory addresses and specify the correct flash type.
2. Appropriately modify the emulation parameter file specified in the -emu option. If Chameleon Debugger is connected to the CPU, some of the parameters returned by the emu command may need to be stored in this file.
3. To verify that SigFlashCmd detects your flash device, run the programmer without the -program and -erase options.
4. Adjust the startup and termination macros used with the -init and -term options, respectively. The initialization macro should enable RAM on the target, set up the PLL, kill the watchdogs, and enable write access to the flash.
5. When testing your macros, use the “echo on” directive to receive screen feedback. Also, utilize the DB, DW, and DD macro commands to validate your settings.

Supported Flash Devices

Table 4 lists the internal and external flash devices supported by Flasher.

D E V I C E	D E S C R I P T I O N
CFI	External CFI flash with any ARM, XScale, or Cortex processor.
STR9	Internal flash for ST Micro STR9x processors (including security bits).
STM32	Internal flash for ST Micro STM32 Cortex-M3 processors.
LPC	LPC2xxx (ARM7-based) and LPC17xx (Cortex-M3 based) processors.
TMPM3	Internal flash for Toshiba TMPM3xx Cortex-M3 processors.
LM3S	Internal flash for Texas Instruments (Luminary Micro) Stellaris Cortex-M3 processors.

DEVICE	DESCRIPTION
SAM7S	Internal flash for Atmel SAM7S processors.
SAM3	Internal flash for Atmel SAM3 processors.
NAND	NAND with selected CPUs and NAND controllers.
RAM	Pseudo-flash in RAM. Enables the Flasher to write to RAM.

TABLE 4 Supported devices.

The type of a flash device must be specified using the `-flash` option. A combination of the `-flash` and `-help` options allows you to display the flash-dependent options for the specified flash programmer. The following sections describe these options in detail.

Programming External CFI NOR Flash

CFI Programmer Options

The figure below shows a fragment of the CFI programmer help information displayed by executing the command: `SigFlashCmd -help -flash CFI`.

```
The CFI programmer options:

  dcc      - Force DCC based programming. Saves time and memory.
  buf      - Force RAM buffer based programming, Requires more memory.
  noblk    - Do not use the programming block buffer.
  blk      - Use the programming block buffer for non-Spansion/AMD devices.
  nounlock - Do not unlock sectors on Intel devices.
  lock     - Lock sectors after programming Intel devices.
```

The `-dcc` and `-buf` options control the communication between the programmer and the CPU. The `-dcc` option uses Debug Communication Channel to pass the data to be programmed without stopping the CPU. The `-buf` option uses the on-target RAM buffer; the CPU is stopped and then restarted between the programmed blocks. Typically, the `-dcc` option—the default for most microprocessors—is faster and requires less memory for the programmer.

The `-noblk` and `-blk` options control the use of the flash on-chip write buffer (block mode programming). Not all flash devices that report block programming as possible support the feature reliably. If this is the case, use the `-noblk` or `-blk` option to force the desired block programming mode.

The `-lock` and `-unlock` options enable locking and unlocking of the programmed sectors, respectively. By default, all affected sectors are unlocked before programming starts and left in this state after the process ends. Flash locking and unlocking are time-consuming operations. You may be able to save time by leaving sectors unlocked when programming for development purposes.

Programming Internal STR9 Flash

STR9 Programmer Options

The figure below shows a fragment of the STR9 programmer help information displayed by executing the command: `SigFlashCmd -help -flash STR9`:

```
The STR9 programmer extra operations (use with the -extra option):

Reading:
?           - Display all settings (except ?mfg).
?idcode    - Display IDCODE as a 32-bit number.
?security   - Display the security bit (0 or 1).
?usercode   - Display USERCODE as a 32-bit number.
?config     - Display CONFIG register as a 64-bit number (with bit names).
?otp        - Display 256 bits of OTP as a hex number.
?mfg        - Display the MFG register as a 32-bit number.

Writing:
usercode=0x11223344 - Program the USERCODE register with a 32-bit number.
config=0xFF00F00... - Program the CONFIG register with up to 64 bits.
security=1          - Set the security bit. Use erase=all to erase it.
erase=config        - Erase the CONFIG register (all 0's).
erase=usercode      - Erase the USERCODE register (all 1's).
erase=all           - Erase all. The only way to erase the security bit.

Writing OTP - *** THESE OPERATIONS CANNOT BE UNDONE - USE WITH CAUTION ***
otp=0xFFFFF1FF...  - Program the OTP bytes (use FF to skip a byte).
otplock=1           - Set the OTPLOCK bit.

CONFIG register's bit names:
MSP0..MSP7          - Bits 0..7   - The Main Sector Protect bits.
SSP0..SSP3          - Bits 32..35 - The Secondary Sector Protect bits.
CSX                  - Bit 48     - The CSx Mapping.
LVD_TH               - Bit 49     - The threshold of the LVD.
LVD_RST_SEL          - Bit 50     - The LVD reset input source.
LVD_WNG_SEL          - Bit 51     - The LVD warning signal input source.
OTP_LOCK             - Bit 63     - The OTP lock bit (cannot be set).

Security:
When the chip is secure, execute the following command to unlock it:

    SigFlashCmd -emu str9.par -flash str9 -extra erase=all
```

See the tests\tr9.log and tests\tr9.bat files for examples of usage.

Note: The one time programmable (OTP) sections of the flash cannot be erased. Use the `extra opt=...` and `extra otplock=1` commands with caution.

Programming Internal LPC Flash

Flasher is invoked when the `-flash LPC` option is used. The programmer supports programming of internal flash in all LPC2xxx, LPC17xx, and LPC11xx devices.

LPC Programmer Specific Files

The following configuration and test files can be used as sample programmer invocations (Table 5):

CONFIGURATION	DESCRIPTION
<code>.ltests\lLPC2xxx.bat</code>	Batch to test the tLPC2xxx programmer.
<code>.ltests\lLPC2xxx.log</code>	Log from running tLPC2xxx.bat with the LPC2138.
<code>.ltests\lLPC17xx.bat</code>	Batch to test the LPC17xx programmer.
<code>.ltests\lLPC17xx.log</code>	Log from running tLPC17xx.bat with the LPC1768.
<code>.ltestdata\vector0_m3.bin</code>	Bootable binary file for the Cortex-M3 processor.
<code>.lboards\lM3swd.par</code>	Emulation config file for the Cortex-M3 SWD interface.
<code>.lboards\lcortex-m0.par</code>	Emulation config file for the Cortex-M0 with SWD interface.

TABLE 5 LPC programmer specific files.

Running the LPC Programmer for the First Time

Before trying to erase or program a device, you must ensure that the programmer can connect to the CPU and access the flash.

In order to connect to the LPC CPU, use the following option:

```
-cpu ARM7TDMI-S          - for LPC2xxx devices
- or -
-cpu Cortex-M3           - for LPC17xx devices
- or -
-cpu Cortex-M0           - for LPC11xx devices
```

One of the easiest ways to connect to the CPU and check if the flash is accessible by interrogating and displaying the flash geometry is illustrated below.

```
C:\Signum\Flasher>SigFlashCmd -cpu ARM7TDMI-S -flash LPC -dump
SigFlashCmd Version 1.15 (C) Signum Systems Corp. 2006-2009
>SigFlashCmd -cpu ARM7TDMI-S -flash LPC -dump
Starting ...
OK - Connected to the emulator ...
OK - Connected to the CPU ...
LPC: Detected CPU LPC2138 (ID=0x0002FF25)
LPC: WARNING: Assuming default crystal frequency 14700KHz
Geometry:
  bank #0 addr=0x00000000 size=32KB (8*4KB)
  bank #0 addr=0x00008000 size=448KB (14*32KB)
  bank #0 addr=0x00078000 size=20KB (5*4KB)
OK - Flash programming finished ...
OK - Exiting ...
OK - Done
```

```
C:\Signum\lib\Debug>SigFlashCmd -cpu Cortex-M3 -flash LPC -dump
SigFlashCmd Version 1.15 (C) Signum Systems Corp. 2006-2009
SigFlashCmd -cpu Cortex-M3 -flash LPC -dump
Starting ...
OK - Connected to the emulator ...
OK - Connected to the CPU ...
LPC: Detected CPU LPC1768 (ID=0x00033F35)
LPC: Using a fixed CPU frequency of 4000KHz
Geometry:
  bank #0 addr=0x00000000 size=64KB (16*4KB)
  bank #0 addr=0x00010000 size=448KB (14*32KB)
OK - Flash programming finished ...
OK - Exiting ...
OK - Done
```

The screen capture above shows two program runs, one for the LPC2138 microcontroller and one for the LPC1768 microcontroller. The programmer reports the CPU names, IDs and flash sector sizes.

Another way to connect to Cortex-M3 based LPC17xx using the SWD protocol. In order to do so, you must use the ARM-SWD probe and apply the following additional command line options:

```
-emu CoresightSWJ=SWD -emu JtagHeader=ARM-SWD
```

- or -

```
-emu M3swd.par
```

The latter method utilizes the ./boards/m3swd.par file that contains the two emulation settings used in the former method.

LPC Programmer Options

The figure below shows a fragment of the LPC programmer help information about the available options and a list of supported processors obtained by executing the command: SigFlashCmd -help -flash LPC.

```
LPC programmer options:

f<freq> - Specify the external crystal frequency in KHz. Default: 14700.
p1      - Handle the User Program Signature in the vector table (default).
p0      - Do not handle the User Program Signature in the vector table.
LPC<cpu> - Select the CPU by name. Default: auto-detection by CPU ID.

Supported CPUs:

LPC1111, LPC1112, LPC1113, LPC1114
LPC1751, LPC1752, LPC1754, LPC1756, LPC1758
LPC1764, LPC1765, LPC1766, LPC1768

LPC2101, LPC2102, LPC2103, LPC2104, LPC2105, LPC2106, LPC2109
LPC2114, LPC2119
```

```
LPC2124, LPC2129
LPC2131, LPC2132, LPC2134, LPC2136, LPC2138
LPC2141, LPC2142, LPC2144, LPC2146, LPC2148
LPC2194
LPC2212, LPC2214
LPC2292, LPC2294
LPC2361, LPC2362, LPC2364, LPC2365, LPC2366, LPC2367, LPC2368
LPC2377, LPC2378
LPC2387, LPC2388
LPC2458, LPC2468, LPC2478
```

The following invocation will program the t.bin file into LPC2xxx device:

```
SigFlashCmd -cpu ARM7TDMI-S -flash LPC,p0,f10000 -program t.bin
```

10MHz external crystal is assumed and User Program Signature will not be handled:

Individual LPC programmer options are discussed below.

Flash Base Address and RAM Usage

The LPC programmer only allows 0x0 as the flash base address. It terminates with an error if a different base address is specified in the -flash LPC@0x100 option.

The LPC programmer always uses internal ram at 0x40000000 (0x10000000 for LPC17xx), and thus it silently ignores the -ram option.

Specifying the Device Manually

After connecting to the CPU successfully, the LPC programmer reads the CPU ID and looks for a device with a matching ID in the programmer's internal database. If the device cannot be not found, the programmer displays an error message and lists the names of supported CPUs.

Different revisions of the LPC2xxx/LPC17xx CPU may have different IDs. If an ID has not been found in the database, it still may be possible to program the CPU. Provided you have the necessary CPU information, you may be able to force the use of the desired CPU by specifying its name with the -flash option, like in -flash LPC,LPCnnnn, where nnnn is the LPC device number, such as 2101 or 7168. Note the comma separator between the programmer name (LPC) and the CPU name (LPCnnnn).

Crystal Frequency

The LPC programmer calls on-chip IAP (In-Application Programming) routines located in the LPC ROM. During erasing or programming, these routines require the actual CPU frequency to be passes to them to ensure proper timing constraints imposed on the flash programming process.

You should specify the f<freq> option as follows:

```
-flash LPC,f12000
```

The default frequency is 14700 KHz.

Experiments show that selecting frequencies higher than that of the actual crystal is not detrimental, otherwise than slowing the programming process to a certain extent. However, due to the possibility of other side effects related to incorrect programming speed, the option should always specify the actual crystal frequency.

Some LPC devices — LPC28xx, LPC17xx for example — have an internal clock oscillator with a fixed frequency. For such devices, the programmer will ignore the specified external crystal frequency and use an internal oscillator during programming. This may generate messages like this:

```
LPC: WARNING: CPU frequency 12000KHz specified by the f option is ignored
LPC: Using fixed CPU frequency 4000KHz
```

User Program Signature

When booting, all LPC devices start to execute the code from internal ROM. After performing a certain amount of housekeeping, the ROM code verifies the integrity of the flash by calculating the sum of the first eight 32-bit words in the flash at 0x0. If this sum is 0x0, then the flash contains valid code and control can be transferred to the user's code in flash memory.

When the programmer attempts to program vectors, it calculates the 32-bit sum and stores it at the unused vector location 0x14 (for LPC2xxx) or 0x1C (for LPC17xx). This is signaled by the following messages:

```
LPC: User Program Signature in file: 0x18171615 (calculated: 0x878E959D)
LPC: New User Program Signature set: 0x878E959D
```

You can skip the calculation of the checksum by specifying the p0 option as follows:

```
-flash LPC,p0
```

The original value from the file is written to the flash when passes the programmer's verification. Otherwise, the following warning message appears:

```
LPC: User Program Signature in file: 0x18171615 (calculated: 0x878E959D)
LPC: WARNING: New User Program Signature not set (p0 option used)
```

The verification process does not compare the calculated checksum with the value stored in the file (which would result in a verification error), which is signaled by the following message:

```
LPC: User Program Signature skipped while verifying ...
```

If your file contains a good checksum — as in a complete flash image saved to a file — and you want the checksum to be verified, use the `-flash LPC,p0` option.

Programming Speed

To display programming times and performance data of a flasher run, use the `-time` option. Table 6 is an example of programming times for a 500 KB file.

CPU AND JTAG TCK	CPU CLOCK	ERASE	PROGRAM	VERIFY OR BLANKCHECK
LPC2138 Adaptive	12 MHz external	0.43 sec	10.91 sec	4.88 sec
LPC2168 Adaptive	48 MHz from 4MHz OSC	0.42 sec	5.97 sec	2.28 sec
LPC1768 JTAG 10Mhz	4MHz internal	0.13 sec	8.95 sec	2.16 sec
LPC1768 JTAG 30Mhz	4MHz internal	0.13 sec	8.17 sec	1.88 sec
LPC1768 SWD 10Mhz	4MHz internal	0.13 sec	12.09 sec	1.73 sec
LPC1768 SWD 30Mhz	4MHz internal	0.13 sec	9.96 sec	1.51 sec

TABLE 6 Timing example for a 500KB image file.

As Table 6 indicates, the erase time for the LPC17xx depends mainly on the type of the CPU. A faster JTAG clock does not affect the erase time, but it does slightly improve the programming and verification speed.

Programming in the SWD mode is somewhat slower than in the JTAG mode. The verify and blankcheck operations are faster, however.

It is expected that future versions of the LPC programmer, especially in the SWD mode, will work even faster,

Programming Internal LM3S Flash

You can use Flasher to program the internal flash of the Luminary Micro Stellaris LM3Sxxxx family of processors.

The command below shows the typical usage of the LM3S programmer.

```
SigFlashCmd -cpu LM3S -flash LM3S -ram 2k@0x2000_0000 -erase -blankcheck -program
myfile.bin -verify
```

Please refer to tests/tLM3S.log and tests/tLM3S.bat for additional usage examples.

LM3S Programmer Options

There are no device specific options for the LM3S flash.

Programming Internal SAM3 FLASH

Use Flasher to program the internal flash on the Atmel AT91SAM3 family of processors.

The command below shows the typical usage of the SAM3 programmer.

```
SigFlashCmd -cpu CORTEX-M3 -flash SAM3 -program myfile.bin -offset 0x0080000 -verify
```

Please refer to tests\tSAM3.log and tests\tSAM3.bat for additional usage examples.

Running the SAM3 Programmer for the First Time

AT91SAM3 devices may contain one or two internal flash devices. One of the devices may be mirrored at 0x0 in addition to its native address. When the flash geometry is displayed using the -dump command, both flashes will be shown as separate banks. If one of the devices is mirrored at 0x0, then the corresponding bank at 0x0 will also be reported.

The figure below shows the geometry of the flash on a SAM3U4E chip. This information is displayed by executing the command: SigFlashCmd -cpu CORTEX-M3 -flash SAM3 -dump

```
SigFlashCmd Version 1.18 (C) Signum Systems Corp. 2006-2010
SigFlashCmd -cpu CORTEX-M3 -flash SAM3 -dump
Starting ...
OK - Connected to the emulator ...
OK - Connected to the CPU ...
    SAM3: Detected SAM3U4E chip
Geometry:
    bank #1 addr=0x00000000 size=128KB (512*256B) ALIAS
    bank #0 addr=0x00080000 size=128KB (512*256B)
    bank #1 addr=0x00100000 size=128KB (512*256B)
OK - Flash programming finished ...
OK - Exiting ...
OK - Done
```

The output indicates the presence of three 128kb regions of flash. In reality, however, there are only two such regions: bank #0 and bank #1. Bank #1 of the flash is accessible at both 0x0 and 0x80000. Note that the region at 0x0 is an “ALIAS”.

The SAM3 chip always boots from 0x0. It is possible to configure it to boot from, flash0, flash1, or ROM.

- If the chip is configured to boot from flash0, then flash0 will be accessible at 0x0 in addition to its native address.
- If the chip is configured to boot from flash1, then flash1 will be accessible at 0x0 in addition to its native address.
- If the chip is configured to boot from ROM, then ROM will be accessible at 0x0 instead of flash.

In order to see what device the chip is currently configured to boot from, use the following command:

```
SigFlashCmd -cpu CORTEX-M3 -flash SAM3 -extra ?BootMode
```

For instructions on how to change to a different boot mode, please see the *Extra Commands* section (p. 17).

To program the chip to boot from code stored in flash0, execute the following commands.

```
SigFlashCmd -cpu CORTEX-M3 -flash SAM3 -extra BootMode=Flash0
```

```
SigFlashCmd -cpu CORTEX-M3 -flash SAM3 -program testdata\vector0_m3.bin
-verify -offset 0x0
```

The first command configures the chip to boot from flash0, and the second one programs the boot code into flash0.

Note the 0x0 offset in the code below. Since flash0 is now the boot device, it is mapped at 0x0. In addition to being mapped at 0x0, flash0 is also accessible at offset 0x80000.

After power cycling the board, the processor should be executing whatever code was in the .bin file programmed.

SAM3 Programmer Options

The figure below shows a fragment of the SAM3 programmer help information displayed by executing the command: `SigFlashCmd -help -flash SAM3`

```
The SAM3 programmer options:
  nounlock  - Do not unlock sectors.
  lock      - Lock sectors after programming.
The SAM3 programmer extra operations (use with the -extra command):
Reading:
  ?BootMode - Display the current boot mode. (ROM or FLASH)
  ?UniqueID - Display the 128bit Unique Identifier.
  ?Lock     - Display the status of the lock bits.
Writing:
  SecurityBit=1      - Program the security bit.
  BootMode=Flash0    - Change chip to boot from flash0.
  BootMode=Flash1    - Change chip to boot from flash1.
  BootMode=ROM       - Change chip to boot from ROM.
```

Unlocking Sectors before Programming

By default, the SAM3 programmer unlocks all of the sectors it needs to access. In order to disable this behavior, run the programmer with the `nounlock` option:

```
SigFlashCMD -cpu CORTEX-M3 -flash SAM3,nounlock
```

Locking Sectors After Programming

The SAM3 programmer does not lock flash sectors after writing data by default. In order to have the programmer lock the sectors it accessed, use the `lock` option:

```
SigFlashCMD -cpu CORTEX-M3 -flash SAM3,lock
```

The `lock` and `nounlock` options may be used in conjunction with each other.

Extra Commands

The `?BootMode` extra command displays the current boot mode. The boot mode can be ROM, Flash0 or Flash1.

```
-extra ?BootMode
```

The `?UniqueID` extra command displays the 128-bit Unique Identifier

```
-extra ?UniqueID
```

The ?Lock extra command displays the lock bits. Each bit that is not zero corresponds to a locked sector.

```
-extra ?Lock
```

The SecurityBit=1 command sets the flash security bit. This bit disables all external read/write access to the flash, and cannot be easily undone. Please refer to the SAM3 user manual before executing this command.

```
-extra SecurityBit=1
```

The BootMode extra command selects the boot mode. It is useful to direct the chip to boot from the code you just programmed into flash.

To boot from Flash0:

```
-extra BootMode=Flash0
```

To boot from Flash1, use the following command (note that not all SAM3 chips have a Flash1).

```
-extra BootMode=Flash1
```

To make the chip boot from internal ROM, use the following extra command.

```
-extra BootMode=ROM
```

The ROM boot mode can be used for UART or USB boot.

Programming Internal STM32 FLASH

Use Flasher to program the internal NOR flash of the ST Microelectronics STM32 family of processors.

The command below shows the typical usage of the STM32 programmer.

```
SigFlashCmd -cpu STM32 -flash STM32,size=512 -ram 32k@0x2000_0000 -erase -
blankcheck -program myfile.bin -offset 0x8000000 -verify
```

Please refer to tests\tSTM32.log and tests\tSTM32.bat for additional usage examples.

STM32 Programmer Options

The figure below shows a fragment of the STM32 programmer help information displayed by executing the command: SigFlashCmd -help -flash STM32

```
The STM32 programmer options:
  size=n - Set the size of the flash chip. n is in KB.
```

The only device specific option that needs to be specified for the STM32 internal flash programmer is the size of the flash. Use the size=n option to specify the flash size in kilobytes.

Programming Internal TMPM3 FLASH

Use Flasher to program the internal NOR flash on the Toshiba TMPM3 family of devices.

The command below shows the typical usage of the TPM3 programmer for the TPM330FDFG board.

```
SigFlashCmd -cpu CORTEX-M3 -flash TPM3,TPM330FDFG -ram 32k@0x2000_0000 -time -
offset -erase -blankcheck -program myfile.bin -verify
```

Please refer to tests\tTPM330.log and tests\tTPM360.log for additional usage examples.

TPM3 Programmer Options

The figure below shows a fragment of the TPM3 programmer help information displayed by executing the command: SigFlashCmd -help -flash TPM3:

```
TPM3 programmer options:

TPM3xxxxxxx - Program the TPM3xxxxxxx device.
              TPM3xxxxxxx is a supported CPU
nounlock     - Do not unlock sectors.
lock         - Lock sectors after programming.

Supported CPUs:

TPM330FDFG, TPM330FYFG, TPM330FWFG,
TPM332FWUG,

TPM360F20FG,
TPM362F10FG,
TPM367FWFG, TPM367FYFG, TPM367FDFG,
TPM368FWFG, TPM368FYFG, TPM368FDFG,
TPM369FYFG, TPM369FDFG,

TPM370FYFG, TPM370FYDFG,
TPM372FWFG,

TPM380FYFG, TPM380FWFG,
TPM382FWFG, TPM382FSFG,

TPM390FWFG,
TPM395FWFG,
```

To program flash on the TPM3, it is necessary to specify exactly which chip is to be programmed. The available chips are listed in the figure above.

Unlocking sectors before programming

By default, the TPM3 programmer unlocks all of the sectors it needs to access. To disable this behavior, run the programmer with the nounlock option:

```
SigFlashCMD -cpu CORTEX-m3 -flash TPM3,TPM3xxxxxxx,nounlock
```

Locking sectors after Programming

By default the TPM3 programmer does not lock flash sectors after writing data. To have the programmer lock the sectors it accessed, use the lock option:

```
SigFlashCMD -cpu CORTEX-m3 -flash TPM3,TMP3xxxxxxxx,lock
```

The lock and nounlock options may be used in conjunction with each other.

Programming NAND Flash

Use Flasher to program external NAND flash. The programmer supports multiple NAND flash devices attached to a limited number of controllers. Please refer to the various tests*NAND.log files for examples of usage.

NAND Programmer Options

The figure below shows a fragment of the NAND programmer help information displayed by executing the command: `SigFlashCmd -help -flash NAND`:

```
The NAND programmer options:
DM644x      - Handle the DaVinci DM644x NAND controller.
DM35x       - Handle the DaVinci DM35x NAND controller.
DM36x       - Handle the DaVinci DM36x controller.
OMAPL13x    - Handle the OMAP-L13x controller.
omap3       - Handle the Omap3 controller.
J2          - Handle the TI's Jacinto2 NAND controller.
iMX27       - Handle the Freescale iMX27 NAND controller.
iMX31       - Handle the Freescale iMX31 NAND controller.
iMX51       - Handle the Freescale iMX51 NAND controller.
S3C2        - Handle the Samsung S3C2xxx NAND controller.

ale=0x<offset> - Set the ALE offset from generic NAND base
cle=0x<offset> - Set the CLE offset from generic NAND base
data=0x<offset> - Set the DATA offset from generic NAND base

linuxbb      - Bbt table must exist at the end of NAND. If not, error.
linuxbb=write - Write a Bbt to NAND.
linuxbb=erase - Allow the Bbt to be erased. Only useful with erasechip
skipbb=0x<offset> - Skip bad block markers starting at <offset>.
failbb       - Fail when bad block is encountered.
maxbb[=n]    - Set the maximum number of bad blocks allowed.
ignorebb     - Ignore bad blocks. Use with caution!

noecc        - Do not do ECC calculation.
swecc        - use software ECC calculation. Supported only on omap3.
spare        - Treat spare bytes as ordinary data. Forces noecc.
swap         - Swap the order of data and spare areas for 2K pages.
boot         - Write boot record signature. For selected controllers only.
```

Choose NAND Controller

The first set of options listed in the figure above, allows you to specify the appropriate NAND flash:

```
-flash NAND, DM644x
```

Generic NAND

Even if your controller (or similar) is not supported, you may be able to program the NAND using the generic NAND configuration. To use this configuration, provide the address of the ale, cle and data lines instead of specifying the controller, as shown in the following example:

```
-flash NAND,ale=0x20000010,cle=0x20000020,data=0x20000030
```

ale is the address of the Address Latch Enable

cle is the address of the Command Latch Enable

data is the data address

Please refer to the board's schematic or reference manual for the details of the NAND chip's connection.

Note: ECC calculation is not supported when using the generic NAND configuration. The programmer can program ECC bytes using the spare option if the bytes are pre-calculated in the data.

Bad Block Management

It is not unusual for a NAND flash to leave the factory with some defective memory blocks. More faulty blocks may develop with use. These bad blocks must be avoided when erasing and programming a NAND flash. The NAND flash programmer offers several ways to manage defective blocks.

The programmer scans the entire NAND in search of bad blocks by default. When found, a bad block is skipped. For instance, if block 5 has a defect, and you attempt to write to it, the programmer will silently write to block 6 instead (or the next good memory block available).

When programming data at an offset, the programmer ignores bad blocks before the offset and skips bad blocks after the offset. On u-boot and Linux, this is the default behavior.

For example, if programming the Linux kernel at 0x400000, bad blocks before 0x400000 are ignored. If there is a bad block at 0x200000, that bad block will not cause the data to shift. Any bad block in the region after 0x400000, where the kernel is to be written, is skipped.

EXAMPLES OF BAD BLOCK RELATED OPTIONS

The Bbt option tells the programmer to require a Bad Block Table (BBT) to exist at the end of NAND. If a BBT does not exist, the programmer will not program the NAND. Some flash file systems store data where the factory bad block marker should be. This makes scanning the flash for bad blocks impossible. If the factory bad block markers have been overwritten, the only way to reliably find bad blocks is to use the BBT.

```
linuxbb - Bbt table must exist at the end of NAND. If not, error.
```

The write option instructs the programmer to scan the chip and write the Bad Block Table in the event the BBT does not currently exist in NAND.

```
linuxbb=write - Write a Bbt to NAND.
```

The erase option allows the Bad Block Table to be erased when using the erasechip command.

```
linuxbb=erase - Allow the Bbt to be erased. Only useful with erasechip.
```

Use the skipbb option to change the address at which bad block skipping begins. If the skipbb=0x<offset> option is not used, the offset is the address at which data is programmed.

```
skipbb=0x<offset> - Skip bad block markers starting at <offset>.
```

The failbb option makes the programmer exit without to erasing, programming or reading a region of the NAND with a bad block in it.

```
failbb - Fail when bad block is encountered.
```

The maxbb option sets the maximum number of bad blocks found, beyond which programming is not attempted. Typically, a large number of bad blocks indicates that either the flash is bad, or the bad block markers have been overwritten. The default value, applied if the maxbb option is used without specifying a value, is 15.

```
maxbb[=n] - Set the maximum number of bad blocks allowed.
```

The ignorebb option tells the NAND flash programmer not to scan for bad blocks and assume that all blocks are good. This option should be used with caution, because it allows bad blocks to be accessed.

```
ignorebb - Ignore bad blocks. Use with caution!
```

Error Correction Code

On all of the controllers explicitly listed, the programmer supports hardware Error Correction Code (ECC) calculation. The number of ECC bytes and their location within the spare area vary from controller to controller. Flasher has several ECC related options.

The noecc option disables ECC calculation. No data is written to the spare area.

```
noecc - Do not do ECC calculation.
```

The swecc option tells the NAND flash programmer to use software ECC calculation. Software ECC uses 3 bytes for every 256 bytes of data. Consequently, for a 2048-byte page, 24 bytes of the spare area are dedicated to ECC. Software ECC is currently only supported on the OMAP3 controller.

```
swecc - use software ECC calculation. Supported only on omap3.
```

The spare option enables reading/writing of the spare area. When this option is used, ECC calculation is disabled.

```
spare - Treat spare bytes as ordinary data. Forces noecc.
```

The swap option tells the programmer that the data on each page is written in the “data spare data spare format,” which may be useful for some older kernel versions. When working with 2K page NAND devices, older versions of the Linux kernel (e.g. 2.6.18) placed 16 spare area bytes after each 512 bytes of data. This lead to overwriting the factory bad block markers. In new versions of the kernel, the problem is alleviated by increasing the spare area placement to 2048 bytes of data followed by 64 bytes spare.

```
swap - Swap the order of data and spare areas for 2K pages.
```

Controller Specific Options

The boot option is specific to the DM644x, DM35x, DM36x, and OMAPL13x controllers.

```
boot - Write boot record signature. For selected controllers only.
```

This family of controller requires a special signature when programming the UBL and UBOOT. Use

```
boot=ubl to program the UBL
boot=uboot to program u-boot
```

The CS option is specific to the iMX51. It allows you to choose a Chip Selects for programming the NAND. The default is CS0.

```
CS<n> - Program NAND connected to Chip Select <n>.
```

You may use any single chip select from CS0 to CS7 or you may use multiple Chip Selects. When using multiple chip selects, you must start with CS0 and utilize 2, 4, or 8 chip selects. See the example below:

```
-flash NAND, iMX51, CS0, CS1
```

This forces the programmer to use both CS0 and CS1 so that the first page of data is written to CS0, the second to CS1, the third to CS0, and so forth.

Programming SPI NOR flash

Flasher can program an external serial NOR flash connected to the Serial Peripheral Interface (SPI). Currently Flasher only supports Atmel Dataflash devices connected to a member of the Freescale iMX51 family of processors.

Please refer to the tests\iMX51SPINOR.bat and iMX51SPINOR.log for files for examples of usage.

SPI NOR Programmer Options

The figure below shows a fragment of the SPI programmer help information displayed by executing the command: SigFlashCmd -help -flash SPI:

```
The SPI programmer options:
iMX51          - Handle the iMX51 SPI controller.
eCSPI1         - Handle flash connected to eCSPI1 on iMX51.
eCSPI2         - Handle flash connected to eCSPI2 on iMX51.

noBinMode      - Allow for programming of non binary page size flash.
noSectorUnlock - Do not disable the sector protection if it is enabled.

The SPI programmer extra operations (use with the -extra command):

?LockDownList - Display list of all permanently locked down sectors.
?SecurityReg   - Read the 128 byte security reg.
SwitchToBin    - Switch the chip to use binary page size.
```

As of this writing, the iMX51 SPI controller is the only device supported. To program the iMX51 SPI, you must use the current version of the Flasher.

```
iMX51 - Handle the iMX51 SPI controller.
```

The programmer is capable of programming an SPI NOR flash connected to either eCSPI1 or eCSPI2 on the iMX51 processor. The correct eCSPI<n> option must be specified, depending on where on the iMX51 CPU the flash is connected.

```
eCSPI1 - Handle flash connected to eCSPI1 on iMX51.
eCSPI2 - Handle flash connected to eCSPI2 on iMX51.
```

Example:

```
SigFlashProgSPI -cpu iMX51 -flash SPI,iMX51,eCSPI1
```

— *or* —

```
SigFlashProgSPI -cpu iMX51 -flash SPI,iMX51,eCSPI2
```

```
noBinMode          - Allow for programming of non binary page size flash.
```

SPI NOR flash leaving the factory has the DataFlash standard page size set to 528 bytes. Such pages are not “Power of 2” addressable. Please refer to the reference manual of your SPI NOR flash for an explanation of binary and non-binary addressing modes.

By default, the SPI programmer does not allow the flash to be programmed if the page sizes are not “Power of 2” binary (256, 512, 1024). Linux does not support non-binary pages.

Use the nobinmode option to allow the programmer to access flash with pages of non-binary sizes.

Example:

```
SigFlashProgSPI -cpu iMX51 -flash SPI,iMX51,eCSPI1,nobinmode
```

The noSectorUnlock option disables the unlocking of previously locked sectors in the SPI NOR flash. If there are any locked sectors, you will not be able to program them. If this option is not applied, the flash programmer unlocks every sector.

```
noSectorUnlock      - Do not disable the sector protection if it is enabled.
```

Example:

```
SigFlashProgSPI -cpu iMX51 -flash SPI,iMX51,eCSPI1,noSectorUnlock
```

The LockDownList extra command lists the sectors that are permanently locked down and thus cannot be programmed.

```
-extra ?LockDownList - Display list of all permanently locked down sectors.
```

Example:

```
SigFlashProgSPI -cpu iMX51 -flash SPI,iMX51,eCSPI1 -extra ?LockDownList
```

The SecurityReg extra command reads the 128 byte security register. The current version of the flash programmer does not have support for programming this register.

```
-extra ?SecurityReg  - Read the 128 byte security reg.
```

Example:

```
SigFlashProgSPI -cpu iMX51 -flash SPI,iMX51,eCSPI1 -extra ?SecurityReg
```

Use the SwitchToBin extra command to switch the SPI NOR flash to binary addressable mode. This operation cannot be undone!

```
-extra SwitchToBin   - Switch the chip to use binary page size.
```

Example:

```
SigFlashProgSPI -cpu iMX51 -flash SPI,iMX51,eCSPI1 -extra ?SwitchToBin
```

The RAM Pseudo-Programmer

The RAM programmer can be used to write to or read data from memory, including a single location, and can be used to verify RAM access.

The programmer treats the entire 32-bit memory address space as one large flash. It allows you to erase a range (filling memory with 0xFF), blank check, program and verify the flash. Be sure to set the -offset (or range for erase and blankcheck) options correctly to avoid writing to non-existent memory. Consider the following example:

```
C:\test>SigFlashCmd -cpu Cortex-M3 -flash RAM -program 2k@0x1000_0000 -verify
SigFlashCmd Version 1.15 (C) Signum Systems Corp. 2006-2009
SigFlashCmd -cpu Cortex-M3 -flash RAM -program 2k@0x1000_0000 -verify
Starting ...
OK - Connected to the emulator ...
OK - Connected to the CPU ...
Programming 0x10000000 .. 0x100007FF (2KB) ...
Verifying 0x10000000 .. 0x100007FF (2KB) ...
OK - Flash programming finished ...
OK - Exiting ...
OK - Done

C:\test>SigFlashCmd -cpu Cortex-M3 -flash RAM -read 16@0x1000_0000
SigFlashCmd Version 1.15 (C) Signum Systems Corp. 2006-2009
SigFlashCmd -cpu Cortex-M3 -flash RAM -read 16@0x1000_0000
Starting ...
OK - Connected to the emulator ...
OK - Connected to the CPU ...
0x10000000 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F 10
OK - Flash programming finished ...
OK - Exiting ...
OK - Done
```

In the preceding example, the first programmer call programs (writes) the 2K of RAM at 0x1000_0000 with a test pattern and verifies the result. The second call reads the initial 16 bytes of the same RAM and displays them.

Programming Multiple Devices

The -connect <SN> option causes SigFlashCmd to connect to the CPU via the JTAGjet emulator with the specified serial number. This makes it possible to program several flash devices in parallel. By running two or more instances of SigFlashCmd, each in its own DOS prompt window, you are able to program multiple devices simultaneously. If the specified emulator cannot be found, SigFlashCmd exits.

In addition, it is possible to create a master batch file that with the use of the DOS start command executes several instances of SigFlashCmd concurrently. This would run each SigFlashCmd in its own window. Unfortunately, the window closes immediately after SigFlashCmd terminates with or without error. It is possible neither to verify that a particular start command has finished running, nor to find out what the resulting error code is. However, with the help of advanced DOS batch programming techniques and by starting multiple batch

files from within one master batch file, you can create a batch program that starts programming multiple devices simultaneously and checks error codes of each individual run.

Custom Flash Devices

Flasher does not provide for customization of flash-dependent modules. The existing flash-dependent SigFlashProg.dll modules handle the majority of external flash devices currently on the market. If you have special needs, please contact Signum Systems. Our contact information can be found at <http://www.signum.com>.



UM-FlashCmd 4.27.11.15.32 780